Cloud Native Essentials

KubeCon + CloudNativeCon 2023 NA: Cloud Native Essentials - A 101 Tutorial to Start Your Cloud Native Journey

Step 1: Welcome to Cloud Native Essentials - A 101 Tutorial to Start Your Cloud Native Journey

Welcome to Cloud Native Essentials Tutorial at KubeCon + CloudNativeCon NA 2023.
In this tutorial we'll give an overview of CNCF graduated projects, what problems they solved, and how to use them. The tutorial will go through the installation, setup, and use of the following: the container runtime containerd to be used with upstream Kubernetes, the latest version of upstream Kubernetes (including etcd), Harbor for a container registry, Helm to deploy an application, Prometheus for monitoring, fluentd for logging, Open Policy Agent and Gatekeeper for admission control for compliance. After this tutorial, you'll be well on your way on your cloud native journey.
We will be using two virtual machines today, **harbor** and **kubernetes** which are located in the tabs in the panel to the right. **kubernetes** will run a Kubernetes cluster, and **harbor** will run a Harbor container registry instance.

Step 2: Containerize an Application: Docker Install

The first step on the Cloud Native Trail Map is Containerization. Let's install Docker, a OCI-compliant container runtime.
Although Docker is not a CNCF project, it was the project that popularized containers. Docker and Docker Compose are required by the Harbor installation in later steps.

Install Docker

1. Run the Docker installer.

```
wget -qO- https://get.docker.com/ | sh
```

*Click to run on **harbor***

Expected output:

```
                                                                    shell
# Executing docker install script, commit: e5543d473431b782227f8908005543bb4389b8de
+ sudo -E sh -c apt-get update -qq >/dev/null
+ sudo -E sh -c DEBIAN_FRONTEND=noninteractive apt-get install -y -qq apt-transport-h
...
```

3. Add user to the docker group.

```
sudo usermod -aG docker $USER
```

*Click to run on **harbor***

4. Activate changes to the docker group.

```
newgrp docker
```

*Click to run on **harbor***

5. Verify Docker is running.

```
docker version
```

*Click to run on **harbor***

Step 3: Containerize an Application: Creating simple-app

Create a simple golang application

1. Since we'll test a simple golang application, install golang.

```
curl -OL https://golang.org/dl/go1.21.3.linux-amd64.tar.gz
```

*Click to run on **harbor***

2. Verify the sha256 checksum.

The sha256 checksum for `go1.21.3.linux-amd64.tar.gz` is `1241381b2843fae5a9707eec1f8fb2ef94d827990582c7c7c32f5bdfbfd420c8` which can be found at https://go.dev/dl.

Verify the sha256 checksum.

```
sha256sum go1.21.3.linux-amd64.tar.gz
echo "1241381b2843fae5a9707eec1f8fb2ef94d827990582c7c7c32f5bdfbfd420c8
go1.21.3.linux-amd64.tar.gz" | sha256sum --check
```

*Click to run on **harbor***

Expected output:

```
go1.21.3.linux-amd64.tar.gz: OK                                    shell
```

3. Extract the tarball to `/usr/local` .

```
sudo tar -C /usr/local -xvf go1.21.3.linux-amd64.tar.gz
```

*Click to run on **harbor***

4. Add go's path to the `.profile` .

```
sed -i -e '$aexport PATH=$PATH:/usr/local/go/bin' ~/.profile
source ~/.profile
```

*Click to run on **harbor***

5. Create the simple golang application.

Create the directory for the golang application.

```
mkdir -p ~/simple-app
```

*Click to run on **harbor***

Create the simple golang application source.

```
cat <<EOF > ~/simple-app/main.go
package main

import (
    "fmt"
    "net/http"
    "os"
    "log"
)

func main() {
    logger := log.New(os.Stdout, "http: ", log.LstdFlags)
    logger.Printf("Server is starting...")
    http.HandleFunc("/", handler)
    http.ListenAndServe(":8080", nil)
}

func handler(w http.ResponseWriter, r *http.Request) {
    hostname, err := os.Hostname()
    if err != nil {
        fmt.Println(err)
        os.Exit(1)
    }
    fmt.Fprintf(w, "Hostname is " + hostname)
}

EOF
```

*Click to run on **harbor***

6. Test the application.

```
go run ~/simple-app/main.go
```

*Click to run on **harbor***

When you see:

```
http: 2023/MM/DD HH:MM:SS Server is starting...
```
shell

7.

Open a browser to test the application http://harbor.${vminfo:harbor:public_ip}.sslip.io:8080.

8.

Stop the program from running with ctrl+c

Step 4: Containerize an Application: Creating a Container Image

Now we have a simple application, let's containerize it using Docker tools.

The **`docker build`** command creates container images using a Dockerfile.

A Dockerfile is a text file with the commands to assemble a container image with a format of **`INSTRUCTION argument`** .

Each Dockerfile starts with the **`FROM`** instruction for the base container image to base the container image from.

1. Create a Dockerfile for the **`simple-app`** .

```
cat <<EOF > ~/simple-app/Dockerfile
FROM golang:1.21-alpine3.17
LABEL project=cloudnativeessentials
WORKDIR /app
COPY *.go ./
RUN go mod init simple-app
RUN go build -o /simple-app
EXPOSE 8080
CMD [ "/simple-app" ]
EOF
```

*Click to run on **harbor***

2. Use **`docker build`** to create the simple-app:0.1 container image.

```
docker build -t simple-app:0.1 ~/simple-app/.
```

*Click to run on **harbor***

Expected output:

```
[+] Building 35.1s (10/10) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 218B
 => [internal] load .dockerignore
 => => transferring context: 2B
```
shell

```
 => [internal] load metadata for docker.io/library/golang:1.21-alpine3.17
 => [1/5] FROM docker.io/library/golang:1.21-alpine3.17@sha256:6e069c2467a974f2b77ebe
 => => resolve docker.io/library/golang:1.21-alpine3.17@sha256:6e069c2467a974f2b77ebe
 => => sha256:a1ef7ce3344464e5090f7a329df2dfd9bb5025b01275806248dba373a82c2413 284.93
 => => sha256:da8cd8fd6e936ae489ee166bf78cbf91a29edeaa0bf7ae97fa2314c7cca723a8 67.02M
 => => sha256:6e069c2467a974f2b77ebee687d820d864e9632d2c606bd7e8478b2be437678e 1.65kB
 => => sha256:06cb1dae8604f1655f089bb3dd210a8123e0c9deaca78033f4629979ed316337 1.16kB
 => => sha256:a9912e40f99f02b22dc040191ae0d85fc96cb4389a8ba9b81657cad47497d818 6.32kB
 => => sha256:9398808236ffac29e60c04ec906d8d409af7fa19dc57d8c65ad167e9c4967006 3.38MB
 => => extracting sha256:9398808236ffac29e60c04ec906d8d409af7fa19dc57d8c65ad167e9c496
 => => sha256:17e1d6a3935137abe8d550b4a284cdd9887a117482e27a1ac714d4993d71d501 155B /
 => => extracting sha256:a1ef7ce3344464e5090f7a329df2dfd9bb5025b01275806248dba373a82c
 => => extracting sha256:da8cd8fd6e936ae489ee166bf78cbf91a29edeaa0bf7ae97fa2314c7cca7
 => => extracting sha256:17e1d6a3935137abe8d550b4a284cdd9887a117482e27a1ac714d4993d71
 => [internal] load build context
 => => transferring context: 29B
 => [2/5] WORKDIR /app
 => [3/5] COPY *.go ./
 => [4/5] RUN go mod init simple-app
 => [5/5] RUN go build -o /simple-app
 => exporting to image
 => => exporting layers
 => => writing image sha256:f261baa9126a6e0fca3bdf9c87ee8d4d5e5f0d8147aa39270bcc23bf9
 => => naming to docker.io/library/simple-app:0.1
```

3. Check if the container image is available locally.

```
docker image ls
```

*Click to run on **harbor***

Expected output:

```
                                                                    shell
REPOSITORY                    TAG       IMAGE ID       CREATED              SIZE
simple-app                    0.1       c451b4bdc00a   About a minute ago   291MB
```

4. Run a container from the new **simple-app:0.1** container image.

```
docker run --name simple-app -p 8080:8080 --detach --rm simple-app:0.1
```

*Click to run on **harbor***

5. Open a browser to test the application. http://harbor.${vminfo:harbor:public_ip}.sslip.io:8080

6. Stop the **simple-app** container.

```
docker stop simple-app
```

*Click to run on **harbor***

Step 5: Install Harbor

Harbor Installation

Prerequisites: Docker Compose and Docker Insecure Registry

You installed Docker in a previous step.

1. Docker Compose is a tool to run a multi-container application. The Harbor installer uses a Docker Compose file to start Harbor.

Install Docker Compose:

```
sudo apt install docker-compose -y
```

*Click to run on **harbor***

Expected output:

```
Reading package lists... Done                                                  shell
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  python3-cached-property python3-docker python3-dockerpty python3-docopt python3-tex
Recommended packages:
  docker.io
The following NEW packages will be installed:
  docker-compose python3-cached-property python3-docker python3-dockerpty python3-doc
0 upgraded, 7 newly installed, 0 to remove and 108 not upgraded.
Need to get 262 kB of archives.
After this operation, 1616 kB of additional disk space will be used.
Get:1 http://us-west-2.ec2.archive.ubuntu.com/ubuntu focal/universe amd64 python3-cac
Get:2 http://us-west-2.ec2.archive.ubuntu.com/ubuntu focal/universe amd64 python3-web
Get:3 http://us-west-2.ec2.archive.ubuntu.com/ubuntu focal/universe amd64 python3-doc
Get:4 http://us-west-2.ec2.archive.ubuntu.com/ubuntu focal/universe amd64 python3-doc
Get:5 http://us-west-2.ec2.archive.ubuntu.com/ubuntu focal/universe amd64 python3-doc
Get:6 http://us-west-2.ec2.archive.ubuntu.com/ubuntu focal/universe amd64 python3-tex
Get:7 http://us-west-2.ec2.archive.ubuntu.com/ubuntu focal/universe amd64 docker-comp
Fetched 262 kB in 0s (8457 kB/s)
Selecting previously unselected package python3-cached-property.
(Reading database ... 62200 files and directories currently installed.)
Preparing to unpack .../0-python3-cached-property_1.5.1-4_all.deb ...
Unpacking python3-cached-property (1.5.1-4) ...
Selecting previously unselected package python3-websocket.
```

```
Preparing to unpack .../1-python3-websocket_0.53.0-2ubuntu1_all.deb ...
Unpacking python3-websocket (0.53.0-2ubuntu1) ...
Selecting previously unselected package python3-docker.
Preparing to unpack .../2-python3-docker_4.1.0-1_all.deb ...
Unpacking python3-docker (4.1.0-1) ...
Selecting previously unselected package python3-dockerpty.
Preparing to unpack .../3-python3-dockerpty_0.4.1-2_all.deb ...
Unpacking python3-dockerpty (0.4.1-2) ...
Selecting previously unselected package python3-docopt.
Preparing to unpack .../4-python3-docopt_0.6.2-2.2ubuntu1_all.deb ...
Unpacking python3-docopt (0.6.2-2.2ubuntu1) ...
Selecting previously unselected package python3-texttable.
Preparing to unpack .../5-python3-texttable_1.6.2-2_all.deb ...
Unpacking python3-texttable (1.6.2-2) ...
Selecting previously unselected package docker-compose.
Preparing to unpack .../6-docker-compose_1.25.0-1_all.deb ...
Unpacking docker-compose (1.25.0-1) ...
Setting up python3-cached-property (1.5.1-4) ...
Setting up python3-texttable (1.6.2-2) ...
Setting up python3-docopt (0.6.2-2.2ubuntu1) ...
Setting up python3-websocket (0.53.0-2ubuntu1) ...
update-alternatives: using /usr/bin/python3-wsdump to provide /usr/bin/wsdump (wsdump
Setting up python3-dockerpty (0.4.1-2) ...
Setting up python3-docker (4.1.0-1) ...
Setting up docker-compose (1.25.0-1) ...
Processing triggers for man-db (2.9.1-1) ...
```

2. In following steps, we will setup Harbor to use http and not https. Docker requires configuration to use insecure registries.

Setup Docker to use an insecure registry by creating an `insecure-registries` entry into `/etc/docker/daemon.json`

Create a `/etc/docker/daemon.json`

```
sudo vi /etc/docker/daemon.json
```

*Click to run on **harbor***

Press `i` for insert mode and enter:

```
{                                                                    shell
    "insecure-registries" : [ "harbor.${vminfo:harbor:public_ip}.sslip.io" ]
}
```

To save: `esc:wq`

3. Restart Docker

```
sudo systemctl daemon-reload
```

*Click to run on **harbor***

```
sudo systemctl restart docker
```

*Click to run on **harbor***

Harbor Installation

4. Download the Harbor online installer

```
wget https://github.com/goharbor/harbor/releases/download/v2.7.3/harbor-online-installer-v2.7.3.tgz
```

*Click to run on **harbor***

Expected output

```
                                                                    shell
--2023-10-12 18:57:59--  https://github.com/goharbor/harbor/releases/download/v2.7.3/
Resolving github.com (github.com)... 192.30.255.113
Connecting to github.com (github.com)|192.30.255.113|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-asset-2e65b
--2023-10-12 18:58:00--  https://objects.githubusercontent.com/github-production-rele
Resolving objects.githubusercontent.com (objects.githubusercontent.com)... 185.199.11
Connecting to objects.githubusercontent.com (objects.githubusercontent.com)|185.199.1
HTTP request sent, awaiting response... 200 OK
Length: 11087 (11K) [application/octet-stream]
Saving to: ‘harbor-online-installer-v2.7.3.tgz’

harbor-online-installer-v2.7.3.tgz    100%[=========================================

2023-10-12 18:58:00 (33.7 MB/s) - ‘harbor-online-installer-v2.7.3.tgz’ saved [11087/1
```

5. Download the asc file

```
wget https://github.com/goharbor/harbor/releases/download/v2.7.3/harbor-online-installer-v2.7.3.tgz.asc
```

*Click to run on **harbor***

Expected output:

```
                                                                    shell
--2023-10-12 18:58:31--  https://github.com/goharbor/harbor/releases/download/v2.7.3/
Resolving github.com (github.com)... 192.30.255.113
Connecting to github.com (github.com)|192.30.255.113|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-asset-2e65b
--2023-10-12 18:58:31--  https://objects.githubusercontent.com/github-production-rele
Resolving objects.githubusercontent.com (objects.githubusercontent.com)... 185.199.10
Connecting to objects.githubusercontent.com (objects.githubusercontent.com)|185.199.1
```

```
HTTP request sent, awaiting response... 200 OK
Length: 833 [application/octet-stream]
Saving to: ‘harbor-online-installer-v2.7.3.tgz.asc’

harbor-online-installer-v2.7.3.tgz.as 100%[=========================================

2023-10-12 18:58:31 (46.2 MB/s) - ‘harbor-online-installer-v2.7.3.tgz.asc’ saved [833
```

6. Obtain the public key for the `asc` file

```
gpg --keyserver hkps://keyserver.ubuntu.com --receive-keys 644FF454C0B4115C
```

*Click to run on **harbor***

Expected output

```
gpg: directory '/home/ubuntu/.gnupg' created                                    shell
gpg: keybox '/home/ubuntu/.gnupg/pubring.kbx' created
gpg: /home/ubuntu/.gnupg/trustdb.gpg: trustdb created
gpg: key 644FF454C0B4115C: public key "Harbor-sign (The key for signing Harbor build)
gpg: Total number processed: 1
gpg:               imported: 1
```

7. Verify the genuity of the package

The `gpg` command verifies that the bundle's signature matches that of the *.asc key file. You should see confirmation that the signature is correct

```
gpg -v --keyserver hkps://keyserver.ubuntu.com --verify harbor-online-installer-
v2.7.3.tgz.asc
```

*Click to run on **harbor***

Expected output

```
gpg: assuming signed data in 'harbor-online-installer-v2.7.3.tgz'              shell
gpg: Signature made Fri Sep  8 03:15:21 2023 UTC
gpg:                using RSA key 7722D168DAEC457806C96FF9644FF454C0B4115C
gpg: using pgp trust model
gpg: Good signature from "Harbor-sign (The key for signing Harbor build) <jiangd@vmwa
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 7722 D168 DAEC 4578 06C9  6FF9 644F F454 C0B4 115C
gpg: binary signature, digest algorithm SHA512, key algorithm rsa4096
```

8. Extract the installer package

```
tar xzvf harbor-online-installer-v2.7.3.tgz
```

*Click to run on **harbor***

Expected output

```
harbor/prepare                                                                  shell
harbor/LICENSE
```

```
harbor/install.sh
harbor/common.sh
harbor/harbor.yml.tmpl
```

Configure the Harbor YAML file

9. Copy the Harbor yaml template to harbor.yml

```
cp ~/harbor/harbor.yml.tmpl ~/harbor/harbor.yml
```

*Click to run on **harbor***

10. Update the Harbor yaml file with your Harbor's url

```
sed -i 's@reg.mydomain.com@harbor.${vminfo:harbor:public_ip}.sslip.io@g' ~/harbor
/harbor.yml
```

*Click to run on **harbor***

11. Update the Harbor yaml file to disable https

```
vi ~/harbor/harbor.yml
```

*Click to run on **harbor***

Edit the `https:` section to read:

```shell
# https:
  # https port for harbor, default is 443
  # port: 443
  # The path of cert and key files for nginx
  # certificate: /your/certificate/path
  # private_key: /your/private/key/path
```

To save: `esc:wq`

Deploy Harbor

12. Run `install.sh`

```
sudo ~/harbor/./install.sh
```

*Click to run on **harbor***

Expected output:

```shell
[Step 0]: checking if docker is installed ...

Note: docker version: 24.0.6

[Step 1]: checking docker-compose is installed ...

Note: Docker Compose version v2.21.0
```

```
[Step 2]: preparing environment ...

[Step 3]: preparing harbor configs ...
prepare base dir is set to /home/ubuntu/harbor
Clearing the configuration file: /config/registryctl/config.yml
...
[+] Running 10/10
 ✔ Network harbor_harbor        Created
 ✔ Container harbor-log         Started
 ✔ Container harbor-db          Started
 ✔ Container redis              Started
 ✔ Container registry           Started
 ✔ Container harbor-portal      Started
 ✔ Container registryctl        Started
 ✔ Container harbor-core        Started
 ✔ Container harbor-jobservice  Started
 ✔ Container nginx              Started
✔ ----Harbor has been installed and started successfully.----
```

10. Check the Harbor containers

```
docker ps
```

*Click to run on **harbor***

Expected output:

```
                                                                                 shell
CONTAINER ID   IMAGE                                COMMAND                  CREATED
bde32f01e01a   goharbor/harbor-jobservice:v2.7.3    "/harbor/entrypoint.…"   57 secon
9c08d0ebaff5   goharbor/nginx-photon:v2.7.3         "nginx -g 'daemon of…"   57 secon
14d6a81db604   goharbor/harbor-core:v2.7.3          "/harbor/entrypoint.…"   57 secon
feddcf259078   goharbor/harbor-db:v2.7.3            "/docker-entrypoint.…"   57 secon
5c125bfedb63   goharbor/registry-photon:v2.7.3      "/home/harbor/entryp…"   57 secon
8ec76866659b   goharbor/redis-photon:v2.7.3         "redis-server /etc/r…"   57 secon
9f60993c55c9   goharbor/harbor-portal:v2.7.3        "nginx -g 'daemon of…"   57 secon
eaa338023837   goharbor/harbor-registryctl:v2.7.3   "/home/harbor/start.…"   57 secon
98e7c9694a13   goharbor/harbor-log:v2.7.3           "/bin/sh -c /usr/loc…"   About a
```

13. Log into Harbor from the Docker client

```
docker login harbor.${vminfo:harbor:public_ip}.sslip.io
```

*Click to run on **harbor***

Use the default admin credentials to login: username: `admin` password: `Harbor12345`

Expected output:

```
                                                                                 shell
```

```
WARNING! Your password will be stored unencrypted in /home/ubuntu/.docker/config.jso
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

14.

Access Harbor at http://harbor.${vminfo:harbor:public_ip}.sslip.io Some browsers might show a warning stating that the Certificate Authority (CA) is unknown. This happens when using a self-signed CA that is not from a trusted third-party CA. You can import the CA to the browser to remove the warning. You can skip this warning. Some Chromium based browsers may not show a skip button. If this is the case, just click anywhere on the error page and type "thisisunsafe" (without quotes). This will force the browser to bypass the warning and accept the certificate.

15.

Use the default admin credentials: username: `admin` password: `Harbor12345`

16.

Create a new project, give it the name `cloudnativeessentials`, set it for public and use the defaults for the rest.

Step 6: Host the Container Image to Harbor

Push your container image to Harbor

1. Use `docker tag` to create an image to include your Harbor instance and cloudnativeessentials project.

```
docker tag simple-app:0.1
harbor.${vminfo:harbor:public_ip}.sslip.io/cloudnativeessentials/simple-app:0.1
```

*Click to run on **harbor***

2. Use `docker push` to push your container image to Harbor

```
docker push harbor.${vminfo:harbor:public_ip}.sslip.io/cloudnativeessentials
/simple-app:0.1
```

*Click to run on **harbor***

Expected output:

```
                                                                        shell
The push refers to repository [harbor.34.223.238.124.sslip.io/cloudnativeessentials/s
508125c76cbd: Pushed
b678f20984d2: Pushed
518b489031a7: Pushed
19b1389586ed: Pushed
ede2f060dece: Pushed
```

```
fe7d12ddfc65: Pushed
fbc321379a11: Pushed
e51777ae0bce: Pushed
2ef3351afa6d: Pushed
5cc3a4df1251: Pushed
2fa37f2ee66e: Pushed
0.1: digest: sha256:593fca85b1a7782af61eaf78cdea99fabeebcb23d8f64fce5d91f77b120d25c8
```

3. Go to the Harbor instance and select the cloudnativeessentials project to verify the container image is present

Pull and run your container image from Harbor

4. Let's remove the existing simple-app container images from our local VM

```
docker image rm simple-app:0.1
docker image rm harbor.${vminfo:harbor:public_ip}.sslip.io/cloudnativeessentials
/simple-app:0.1
```

*Click to run on **harbor***

5. Use `docker run` to pull the `simple-app:0.1` container image from your Harbor instance and run a container

```
docker run --name simple-app -p 8080:8080 --detach --rm
harbor.${vminfo:harbor:public_ip}.sslip.io/cloudnativeessentials/simple-app:0.1
```

*Click to run on **harbor***

6.

Open a browser to test the application http://harbor.${vminfo:harbor:public_ip}.sslip.io:8080

7.

Stop the `simple-app` container

```
docker stop simple-app
```

*Click to run on **harbor***

Step 7: Kubernetes Requirements

Kubernetes Requirements: cgroups v2, iptables, socat, conntrack

cgroups v2

Starting in Kubernetes 1.28, Kubernetes can leverage cgroups v2 for enhanced resource management

capabilities including swap. If cgroups v1 is used then you would have to disable swap with `sudo swapoff -a` and disable swap in `/etc/fstab` or `systemd.swap` .

1. Check if cgroups v2 is used.

```
stat -fc %T /sys/fs/cgroup/
```

*Click to run on **kubernetes***

Expected output:

```
cgroup2fs                                                    shell
```

If the output is `tempfs` then cgroups v1 is used.

Forward IPv4 and iptables bridged traffic

2. Forwarding IPv4 and letting iptables see bridged traffic. Load the `overlay` and `br_netfilter` kernel modules in the current environment and set them to load on boot

```
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF
```

*Click to run on **kubernetes***

3. The `modprobe` command can add or remove kernel modules. Use `modprobe` to add the `overlay` and `br_netfilter` kernel modules. The `br_netfilter` module enables transparent masquerading and facilitate Virtual Extensible LAN (VxLAN) traffic for communication between Kubernetes pods.

```
sudo modprobe overlay
```

*Click to run on **kubernetes***

```
sudo modprobe br_netfilter
```

*Click to run on **kubernetes***

4. Configure sysctl parameters to persist reboots

```
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables  = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward                 = 1
EOF
```

*Click to run on **kubernetes***

5. Apply sysctl parameters without rebooting

```
sudo sysctl --system
```

*Click to run on **kubernetes***

Expected output:

```shell
* Applying /etc/sysctl.d/10-console-messages.conf ...
kernel.printk = 4 4 1 7
* Applying /etc/sysctl.d/10-ipv6-privacy.conf ...
net.ipv6.conf.all.use_tempaddr = 2
net.ipv6.conf.default.use_tempaddr = 2
* Applying /etc/sysctl.d/10-kernel-hardening.conf ...
kernel.kptr_restrict = 1
* Applying /etc/sysctl.d/10-magic-sysrq.conf ...
kernel.sysrq = 176
* Applying /etc/sysctl.d/10-network-security.conf ...
net.ipv4.conf.default.rp_filter = 2
net.ipv4.conf.all.rp_filter = 2
* Applying /etc/sysctl.d/10-ptrace.conf ...
kernel.yama.ptrace_scope = 1
* Applying /etc/sysctl.d/10-zeropage.conf ...
vm.mmap_min_addr = 65536
* Applying /usr/lib/sysctl.d/50-default.conf ...
kernel.core_uses_pid = 1
net.ipv4.conf.default.rp_filter = 2
net.ipv4.conf.default.accept_source_route = 0
sysctl: setting key "net.ipv4.conf.all.accept_source_route": Invalid argument
net.ipv4.conf.default.promote_secondaries = 1
sysctl: setting key "net.ipv4.conf.all.promote_secondaries": Invalid argument
net.ipv4.ping_group_range = 0 2147483647
net.core.default_qdisc = fq_codel
fs.protected_hardlinks = 1
fs.protected_symlinks = 1
fs.protected_regular = 1
fs.protected_fifos = 1
* Applying /usr/lib/sysctl.d/50-pid-max.conf ...
kernel.pid_max = 4194304
* Applying /etc/sysctl.d/99-cloudimg-ipv6.conf ...
net.ipv6.conf.all.use_tempaddr = 0
net.ipv6.conf.default.use_tempaddr = 0
* Applying /usr/lib/sysctl.d/99-protect-links.conf ...
fs.protected_fifos = 1
fs.protected_hardlinks = 1
fs.protected_regular = 2
fs.protected_symlinks = 1
* Applying /etc/sysctl.d/99-sysctl.conf ...
* Applying /etc/sysctl.d/k8s.conf ...
net.bridge.bridge-nf-call-iptables = 1
net.bridge.bridge-nf-call-ip6tables = 1
```

```
net.ipv4.ip_forward = 1
* Applying /etc/sysctl.conf ...
```

6. Verify that the br_netfilter, overlay modules are loaded

```
lsmod | grep br_netfilter
```

*Click to run on **kubernetes***

Expected output:

```
br_netfilter           32768  0                                                shell
bridge                331776  1 br_netfilter
```

```
lsmod | grep overlay
```

*Click to run on **kubernetes***

Expected output:

```
overlay               159744  0                                                shell
```

7. Verify that the net.bridge.bridge-nf-call-iptables, net.bridge.bridge-nf-call-ip6tables, and net.ipv4.ip_forward system variables are set to 1 in your sysctl config by running the following command:

```
sysctl net.bridge.bridge-nf-call-iptables net.bridge.bridge-nf-call-ip6tables
net.ipv4.ip_forward
```

*Click to run on **kubernetes***

Expected output:

```
net.bridge.bridge-nf-call-iptables = 1                                         shell
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward = 1
```

8. socat helps redirect traffic within the Kubernetes cluster. Install socat

```
sudo apt install socat -y
```

*Click to run on **kubernetes***

Expected output:

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  socat
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 349 kB of archives.
After this operation, 1381 kB of additional disk space will be used.
Get:1 http://us-west-2.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 socat amd64
1.7.4.1-3ubuntu4 [349 kB]
```

```
Fetched 349 kB in 0s (18.2 MB/s)
Selecting previously unselected package socat.
(Reading database ... 64295 files and directories currently installed.)
Preparing to unpack .../socat_1.7.4.1-3ubuntu4_amd64.deb ...
Unpacking socat (1.7.4.1-3ubuntu4) ...
Setting up socat (1.7.4.1-3ubuntu4) ...
Processing triggers for man-db (2.10.2-1) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
```

8. conntrack helps connection information between Pods and Services. Install conntrack

```
sudo apt install conntrack -y
```

*Click to run on **kubernetes***

Expected output:

```
Reading package lists... Done                                                  shell
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  conntrack
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 33.5 kB of archives.
After this operation, 104 kB of additional disk space will be used.
Get:1 http://us-west-2.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 conntrack amd6
Fetched 33.5 kB in 0s (1198 kB/s)
Selecting previously unselected package conntrack.
(Reading database ... 64335 files and directories currently installed.)
Preparing to unpack .../conntrack_1%3a1.4.6-2build2_amd64.deb ...
Unpacking conntrack (1:1.4.6-2build2) ...
Setting up conntrack (1:1.4.6-2build2) ...
Processing triggers for man-db (2.10.2-1) ...
Scanning processes...
```

```
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
```

Step 8: Kubernetes Installation: containerd

Kubernetes Preqeuisities: containerd

Since Kubernetes orchestrates containers, we need a container runtime. Kubernetes uses the Container Runtime Interface (CRI) to interface with a container runtime therefore the container runtime needs to be CRI compliant and OCI compliant. There are several to choose from, we will use containerd, a CNCF graduated project.

Install containerd

1. Download the containerd archive from https://github.com/containerd/containerd/releases, verify its sha256sum, and extract it under `/usr/local` .

```
wget https://github.com/containerd/containerd/releases/download/v1.7.7/containerd-1.7.7-linux-amd64.tar.gz
```

*Click to run on **kubernetes***

Expected output:

```
                                                                    shell
--2023-10-17 20:08:23--  https://github.com/containerd/containerd/releases/download/v
Resolving github.com (github.com)... 192.30.255.112
Connecting to github.com (github.com)|192.30.255.112|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-asset-2e65b
--2023-10-17 20:08:24--  https://objects.githubusercontent.com/github-production-rele
Resolving objects.githubusercontent.com (objects.githubusercontent.com)... 185.199.10
Connecting to objects.githubusercontent.com (objects.githubusercontent.com)|185.199.1
HTTP request sent, awaiting response... 200 OK
Length: 46887113 (45M) [application/octet-stream]
Saving to: ‘containerd-1.7.7-linux-amd64.tar.gz’
```

```
containerd-1.7.7-linux-amd64.tar.gz   100%[=========================================

2023-10-17 20:08:25 (60.0 MB/s) - ‘containerd-1.7.7-linux-amd64.tar.gz’ saved [46887
```

2. Download the sha256sum.

```
wget https://github.com/containerd/containerd/releases/download/v1.7.7/containerd-
1.7.7-linux-amd64.tar.gz.sha256sum
```

*Click to run on **kubernetes***

Expected output:

```
                                                                                shell
--2023-10-17 20:08:51--  https://github.com/containerd/containerd/releases/download/v
Resolving github.com (github.com)... 192.30.255.113
Connecting to github.com (github.com)|192.30.255.113|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-asset-2e65b
--2023-10-17 20:08:51--  https://objects.githubusercontent.com/github-production-rele
Resolving objects.githubusercontent.com (objects.githubusercontent.com)... 185.199.10
Connecting to objects.githubusercontent.com (objects.githubusercontent.com)|185.199.1
HTTP request sent, awaiting response... 200 OK
Length: 102 [application/octet-stream]
Saving to: ‘containerd-1.7.7-linux-amd64.tar.gz.sha256sum’

containerd-1.7.7-linux-amd64.tar.gz.s 100%[=========================================

2023-10-17 20:08:51 (5.96 MB/s) - ‘containerd-1.7.7-linux-amd64.tar.gz.sha256sum’ sav
```

3. Verify the sha256 checksum

```
sha256sum -c containerd-1.7.7-linux-amd64.tar.gz.sha256sum
```

*Click to run on **kubernetes***

Expected output:

```
containerd-1.7.7-linux-amd64.tar.gz: OK                                         shell
```

4. Extract the containerd archive into `/usr/local`

```
sudo tar Cxzvf /usr/local containerd-1.7.7-linux-amd64.tar.gz
```

*Click to run on **kubernetes***

Expected output:

```
bin/                                                                            shell
bin/containerd-shim-runc-v1
bin/containerd-stress
bin/containerd-shim-runc-v2
bin/containerd
```

```
bin/containerd-shim
bin/ctr
```

4. Since we will use systemd to containerd, download the `containerd.service` unit file.

```
sudo wget -P /usr/local/lib/systemd/system https://raw.githubusercontent.com
/containerd/containerd/main/containerd.service
```

*Click to run on **kubernetes***

5. Reload systemd manager configuration to reload all unit files.

```
sudo systemctl daemon-reload
```

*Click to run on **kubernetes***

6. Enable the containerd service.

```
sudo systemctl enable --now containerd
```

*Click to run on **kubernetes***

7. Install runc, verify its sha256sum, and install it as /usr/local/sbin/run

Download runc

```
wget https://github.com/opencontainers/runc/releases/download/v1.1.9/runc.amd64
```

*Click to run on **kubernetes***

Expected output:

```
                                                                        shell
--2023-10-17 20:15:07--  https://github.com/opencontainers/runc/releases/download/v1.
Resolving github.com (github.com)... 192.30.255.113
Connecting to github.com (github.com)|192.30.255.113|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-asset-2e65b
--2023-10-17 20:15:08--  https://objects.githubusercontent.com/github-production-rele
Resolving objects.githubusercontent.com (objects.githubusercontent.com)... 185.199.10
Connecting to objects.githubusercontent.com (objects.githubusercontent.com)|185.199.1
HTTP request sent, awaiting response... 200 OK
Length: 10684992 (10M) [application/octet-stream]
Saving to: ‘runc.amd64’

runc.amd64                          100%[=======================================

2023-10-17 20:15:08 (93.0 MB/s) - ‘runc.amd64’ saved [10684992/10684992]
```

Download the asc file.

```
wget https://github.com/opencontainers/runc/releases/download/v1.1.9
/runc.amd64.asc
```

*Click to run on **kubernetes***

Expected output:

```
                                                                                shell
--2023-10-17 20:15:45--  https://github.com/opencontainers/runc/releases/download/v1
Resolving github.com (github.com)... 192.30.255.112
Connecting to github.com (github.com)|192.30.255.112|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-asset-2e65b
--2023-10-17 20:15:46--  https://objects.githubusercontent.com/github-production-rele
Resolving objects.githubusercontent.com (objects.githubusercontent.com)... 185.199.11
Connecting to objects.githubusercontent.com (objects.githubusercontent.com)|185.199.1
HTTP request sent, awaiting response... 200 OK
Length: 854 [application/octet-stream]
Saving to: ‘runc.amd64.asc’

runc.amd64.asc                      100%[=======================================

2023-10-17 20:15:46 (46.0 MB/s) - ‘runc.amd64.asc’ saved [854/854]
```

8. Install runc

```
sudo install -m 755 runc.amd64 /usr/local/sbin/runc
```

*Click to run on **kubernetes***

9. containerd uses a configuration file called `config.toml` to specify dameon-level options. The `config.toml` is located `/etc/containerd/config.toml`

Generate the default `config.toml`

```
sudo mkdir -p /etc/containerd
```

*Click to run on **kubernetes***

```
containerd config default | sudo tee /etc/containerd/config.toml
```

*Click to run on **kubernetes***

Expected output:

```
disabled_plugins = []                                                           shell
imports = []
oom_score = 0
plugin_dir = ""
required_plugins = []
root = "/var/lib/containerd"
state = "/run/containerd"
temp = ""
version = 2

[cgroup]
  path = ""
```

```
[debug]
  address = ""
  format = ""
  gid = 0
  level = ""
  uid = 0

[grpc]
  address = "/run/containerd/containerd.sock"
  gid = 0
  max_recv_message_size = 16777216
  max_send_message_size = 16777216
  tcp_address = ""
  tcp_tls_ca = ""
  tcp_tls_cert = ""
  tcp_tls_key = ""
  uid = 0

[metrics]
  address = ""
  grpc_histogram = false

[plugins]

  [plugins."io.containerd.gc.v1.scheduler"]
    deletion_threshold = 0
    mutation_threshold = 100
    pause_threshold = 0.02
    schedule_delay = "0s"
    startup_delay = "100ms"

  [plugins."io.containerd.grpc.v1.cri"]
    cdi_spec_dirs = ["/etc/cdi", "/var/run/cdi"]
    device_ownership_from_security_context = false
    disable_apparmor = false
    disable_cgroup = false
    disable_hugetlb_controller = true
    disable_proc_mount = false
    disable_tcp_service = true
    drain_exec_sync_io_timeout = "0s"
    enable_cdi = false
    enable_selinux = false
    enable_tls_streaming = false
    enable_unprivileged_icmp = false
    enable_unprivileged_ports = false
```

```
ignore_image_defined_volumes = false
image_pull_progress_timeout = "1m0s"
max_concurrent_downloads = 3
max_container_log_line_size = 16384
netns_mounts_under_state_dir = false
restrict_oom_score_adj = false
sandbox_image = "registry.k8s.io/pause:3.8"
selinux_category_range = 1024
stats_collect_period = 10
stream_idle_timeout = "4h0m0s"
stream_server_address = "127.0.0.1"
stream_server_port = "0"
systemd_cgroup = false
tolerate_missing_hugetlb_controller = true
unset_seccomp_profile = ""

[plugins."io.containerd.grpc.v1.cri".cni]
  bin_dir = "/opt/cni/bin"
  conf_dir = "/etc/cni/net.d"
  conf_template = ""
  ip_pref = ""
  max_conf_num = 1
  setup_serially = false

[plugins."io.containerd.grpc.v1.cri".containerd]
  default_runtime_name = "runc"
  disable_snapshot_annotations = true
  discard_unpacked_layers = false
  ignore_blockio_not_enabled_errors = false
  ignore_rdt_not_enabled_errors = false
  no_pivot = false
  snapshotter = "overlayfs"

  [plugins."io.containerd.grpc.v1.cri".containerd.default_runtime]
    base_runtime_spec = ""
    cni_conf_dir = ""
    cni_max_conf_num = 0
    container_annotations = []
    pod_annotations = []
    privileged_without_host_devices = false
    privileged_without_host_devices_all_devices_allowed = false
    runtime_engine = ""
    runtime_path = ""
    runtime_root = ""
    runtime_type = ""
```

```
    sandbox_mode = ""
    snapshotter = ""

    [plugins."io.containerd.grpc.v1.cri".containerd.default_runtime.options]

  [plugins."io.containerd.grpc.v1.cri".containerd.runtimes]

    [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc]
      base_runtime_spec = ""
      cni_conf_dir = ""
      cni_max_conf_num = 0
      container_annotations = []
      pod_annotations = []
      privileged_without_host_devices = false
      privileged_without_host_devices_all_devices_allowed = false
      runtime_engine = ""
      runtime_path = ""
      runtime_root = ""
      runtime_type = "io.containerd.runc.v2"
      sandbox_mode = "podsandbox"
      snapshotter = ""

      [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.options]
        BinaryName = ""
        CriuImagePath = ""
        CriuPath = ""
        CriuWorkPath = ""
        IoGid = 0
        IoUid = 0
        NoNewKeyring = false
        NoPivotRoot = false
        Root = ""
        ShimCgroup = ""
        SystemdCgroup = false

  [plugins."io.containerd.grpc.v1.cri".containerd.untrusted_workload_runtime]
    base_runtime_spec = ""
    cni_conf_dir = ""
    cni_max_conf_num = 0
    container_annotations = []
    pod_annotations = []
    privileged_without_host_devices = false
    privileged_without_host_devices_all_devices_allowed = false
    runtime_engine = ""
    runtime_path = ""
```

```
      runtime_root = ""
      runtime_type = ""
      sandbox_mode = ""
      snapshotter = ""

      [plugins."io.containerd.grpc.v1.cri".containerd.untrusted_workload_runtime.op

  [plugins."io.containerd.grpc.v1.cri".image_decryption]
    key_model = "node"

  [plugins."io.containerd.grpc.v1.cri".registry]
    config_path = ""

    [plugins."io.containerd.grpc.v1.cri".registry.auths]

    [plugins."io.containerd.grpc.v1.cri".registry.configs]

    [plugins."io.containerd.grpc.v1.cri".registry.headers]

    [plugins."io.containerd.grpc.v1.cri".registry.mirrors]

  [plugins."io.containerd.grpc.v1.cri".x509_key_pair_streaming]
    tls_cert_file = ""
    tls_key_file = ""

[plugins."io.containerd.internal.v1.opt"]
  path = "/opt/containerd"

[plugins."io.containerd.internal.v1.restart"]
  interval = "10s"

[plugins."io.containerd.internal.v1.tracing"]
  sampling_ratio = 1.0
  service_name = "containerd"

[plugins."io.containerd.metadata.v1.bolt"]
  content_sharing_policy = "shared"

[plugins."io.containerd.monitor.v1.cgroups"]
  no_prometheus = false

[plugins."io.containerd.nri.v1.nri"]
  disable = true
  disable_connections = false
  plugin_config_path = "/etc/nri/conf.d"
```

```
    plugin_path = "/opt/nri/plugins"
    plugin_registration_timeout = "5s"
    plugin_request_timeout = "2s"
    socket_path = "/var/run/nri/nri.sock"

  [plugins."io.containerd.runtime.v1.linux"]
    no_shim = false
    runtime = "runc"
    runtime_root = ""
    shim = "containerd-shim"
    shim_debug = false

  [plugins."io.containerd.runtime.v2.task"]
    platforms = ["linux/amd64"]
    sched_core = false

  [plugins."io.containerd.service.v1.diff-service"]
    default = ["walking"]

  [plugins."io.containerd.service.v1.tasks-service"]
    blockio_config_file = ""
    rdt_config_file = ""

  [plugins."io.containerd.snapshotter.v1.aufs"]
    root_path = ""

  [plugins."io.containerd.snapshotter.v1.blockfile"]
    fs_type = ""
    mount_options = []
    root_path = ""
    scratch_file = ""

  [plugins."io.containerd.snapshotter.v1.btrfs"]
    root_path = ""

  [plugins."io.containerd.snapshotter.v1.devmapper"]
    async_remove = false
    base_image_size = ""
    discard_blocks = false
    fs_options = ""
    fs_type = ""
    pool_name = ""
    root_path = ""

  [plugins."io.containerd.snapshotter.v1.native"]
```

```
    root_path = ""

  [plugins."io.containerd.snapshotter.v1.overlayfs"]
    mount_options = []
    root_path = ""
    sync_remove = false
    upperdir_label = false

  [plugins."io.containerd.snapshotter.v1.zfs"]
    root_path = ""

  [plugins."io.containerd.tracing.processor.v1.otlp"]
    endpoint = ""
    insecure = false
    protocol = ""

  [plugins."io.containerd.transfer.v1.local"]
    config_path = ""
    max_concurrent_downloads = 3
    max_concurrent_uploaded_layers = 3

    [[plugins."io.containerd.transfer.v1.local".unpack_config]]
      differ = ""
      platform = "linux/amd64"
      snapshotter = "overlayfs"

[proxy_plugins]

[stream_processors]

  [stream_processors."io.containerd.ocicrypt.decoder.v1.tar"]
    accepts = ["application/vnd.oci.image.layer.v1.tar+encrypted"]
    args = ["--decryption-keys-path", "/etc/containerd/ocicrypt/keys"]
    env = ["OCICRYPT_KEYPROVIDER_CONFIG=/etc/containerd/ocicrypt/ocicrypt_keyprovider
    path = "ctd-decoder"
    returns = "application/vnd.oci.image.layer.v1.tar"

  [stream_processors."io.containerd.ocicrypt.decoder.v1.tar.gzip"]
    accepts = ["application/vnd.oci.image.layer.v1.tar+gzip+encrypted"]
    args = ["--decryption-keys-path", "/etc/containerd/ocicrypt/keys"]
    env = ["OCICRYPT_KEYPROVIDER_CONFIG=/etc/containerd/ocicrypt/ocicrypt_keyprovider
    path = "ctd-decoder"
    returns = "application/vnd.oci.image.layer.v1.tar+gzip"

[timeouts]
```

```
  "io.containerd.timeout.bolt.open" = "0s"
  "io.containerd.timeout.metrics.shimstats" = "2s"
  "io.containerd.timeout.shim.cleanup" = "5s"
  "io.containerd.timeout.shim.load" = "5s"
  "io.containerd.timeout.shim.shutdown" = "3s"
  "io.containerd.timeout.task.state" = "2s"

[ttrpc]
  address = ""
  gid = 0
  uid = 0
```

10. Set the `systemd` cgroup driver with runc Set the `SystemdCgroup` setting to true

```
sudo sed -i 's@SystemdCgroup = false@SystemdCgroup = true@g' /etc/containerd
/config.toml
```

*Click to run on **kubernetes***

11. Restart containerd

```
sudo systemctl restart containerd
```

*Click to run on **kubernetes***

Step 9: Kubernetes Installation: cni and crictl

CNI installation

CNI plugins are required to implement the Kubernetes network model.

1. Install CNI plugins.

```
CNI_PLUGINS_VERSION="v1.3.0"
ARCH="amd64"
DEST="/opt/cni/bin"
```

*Click to run on **kubernetes***

```
sudo mkdir -p "$DEST"
```

*Click to run on **kubernetes***

```
curl -L "https://github.com/containernetworking/plugins/releases/download
/${CNI_PLUGINS_VERSION}/cni-plugins-linux-${ARCH}-${CNI_PLUGINS_VERSION}.tgz" |
sudo tar -C "$DEST" -xz
```

*Click to run on **kubernetes***

Expected output:

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
```

```
                                 Dload  Upload   Total   Spent    Left  Speed shell
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 43.2M  100 43.2M    0     0  25.0M      0  0:00:01  0:00:01 --:--:-- 43.0M
```

Kubernetes Prerequisites: crictl

`crictl` is a CLI for CRI-compatible container runtimes. **`crictl`** is required for kubeadm and the kubelet container runtime interface

2. Install **`crictl`** :

```
DOWNLOAD_DIR="/usr/local/bin"
```

*Click to run on **kubernetes***

```
sudo mkdir -p "$DOWNLOAD_DIR"
```

*Click to run on **kubernetes***

```
CRICTL_VERSION="v1.28.0"
```

*Click to run on **kubernetes***

```
ARCH="amd64"
```

*Click to run on **kubernetes***

```
curl -L "https://github.com/kubernetes-sigs/cri-tools/releases/download
/${CRICTL_VERSION}/crictl-${CRICTL_VERSION}-linux-${ARCH}.tar.gz" | sudo tar -C
$DOWNLOAD_DIR -xz
```

*Click to run on **kubernetes***

Expected output:

```
                                                                      shell
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 22.7M  100 22.7M    0     0  12.9M      0  0:00:01  0:00:01 --:--:-- 40.4M
```

2. Test crictl

```
crictl --version
```

*Click to run on **kubernetes***

Expected output:

```
crictl version v1.28.0                                                shell
```

Step 10: Kubernetes Installation: kubeadm and kubelet

Kubernetes Installation: kubeadm, kubelet, kubelet systemd service

1. Check the latest Kubernetes version.

```
curl -sSL https://dl.k8s.io/release/stable.txt -w "\n"
```

*Click to run on **kubernetes***

2. Install kubeadm, kubelet, and kubelet systemd service.

```
RELEASE="$(curl -sSL https://dl.k8s.io/release/stable.txt)"
ARCH="amd64"
RELEASE_VERSION="v0.16.2"
```

*Click to run on **kubernetes***

```
cd $DOWNLOAD_DIR
sudo curl -L --remote-name-all https://dl.k8s.io/release/${RELEASE}/bin/linux
/${ARCH}/{kubeadm,kubelet}
```

*Click to run on **kubernetes***

Make kubeadm and kubelet executable.

```
sudo chmod +x {kubeadm,kubelet}
```

*Click to run on **kubernetes***

Download the kubelet systemd unit file.

```
curl -sSL "https://raw.githubusercontent.com/kubernetes/release/${RELEASE_VERSION}
/cmd/krel/templates/latest/kubelet/kubelet.service" | sed "s:/usr
/bin:${DOWNLOAD_DIR}:g" | sudo tee /etc/systemd/system/kubelet.service
```

*Click to run on **kubernetes***

```
sudo mkdir -p /etc/systemd/system/kubelet.service.d
```

*Click to run on **kubernetes***

The 10-kubeadm.conf file is the configuration for how systemd should run the kubelet. Download the 10-kubeadm.conf file.

```
curl -sSL "https://raw.githubusercontent.com/kubernetes/release/${RELEASE_VERSION}
/cmd/krel/templates/latest/kubeadm/10-kubeadm.conf" | sed "s:/usr
/bin:${DOWNLOAD_DIR}:g" | sudo tee /etc/systemd/system/kubelet.service.d/10-
kubeadm.conf
cd ~
```

*Click to run on **kubernetes***

Expected output:

```
                                                                                shell
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   138  100   138    0     0    743      0 --:--:-- --:--:-- --:--:--   745
100 48.4M  100 48.4M    0     0  23.7M      0  0:00:02  0:00:02 --:--:-- 31.7M
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   138  100   138    0     0   2432      0 --:--:-- --:--:-- --:--:--  2464
100  105M  100  105M    0     0   156M      0 --:--:-- --:--:-- --:--:--  156M
[Unit]
Description=kubelet: The Kubernetes Node Agent
Documentation=https://kubernetes.io/docs/
Wants=network-online.target
After=network-online.target

[Service]
ExecStart=/usr/local/bin/kubelet
Restart=always
StartLimitInterval=0
RestartSec=10

[Install]
WantedBy=multi-user.target
# Note: This dropin only works with kubeadm and kubelet v1.11+
[Service]
Environment="KUBELET_KUBECONFIG_ARGS=--bootstrap-kubeconfig=/etc/kubernetes/bootstrap
Environment="KUBELET_CONFIG_ARGS=--config=/var/lib/kubelet/config.yaml"
# This is a file that "kubeadm init" and "kubeadm join" generates at runtime, populat
EnvironmentFile=-/var/lib/kubelet/kubeadm-flags.env
# This is a file that the user can use for overrides of the kubelet args as a last re
# the .NodeRegistration.KubeletExtraArgs object in the configuration files instead. K
EnvironmentFile=-/etc/sysconfig/kubelet
ExecStart=
ExecStart=/usr/local/bin/kubelet $KUBELET_KUBECONFIG_ARGS $KUBELET_CONFIG_ARGS $KUBEL
```

3. Enable and start kubelet

```
sudo systemctl enable --now kubelet
```

*Click to run on **kubernetes***

Expected output:

```
                                                                                shell
Created symlink /etc/systemd/system/multi-user.target.wants/kubelet.service → /etc/sy
```

Note: The kubelet is crashlooping while waiting for kubeadm

Step 11: Kubernetes Installation: kubectl

Install kubectl

kubectl is the Kubernetes cli tool and allows you to run commands against Kubernetes clusters.

1. Download the kubectl binary for the latest release

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release
/stable.txt)/bin/linux/amd64/kubectl"
```

*Click to run on **kubernetes***

Expected output:

```
                                                                        shell
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   138  100   138    0     0    774      0 --:--:-- --:--:-- --:--:--   775
100 47.5M  100 47.5M    0     0  48.0M      0 --:--:-- --:--:-- --:--:--  114M
```

2. Download the kubectl checksum file

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release
/stable.txt)/bin/linux/amd64/kubectl.sha256"
```

*Click to run on **kubernetes***

Expected output:

```
                                                                        shell
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   138  100   138    0     0    831      0 --:--:-- --:--:-- --:--:--   836
100    64  100    64    0     0    219      0 --:--:-- --:--:-- --:--:--   219
```

3. Validate the kubectl binary with the checksum file

```
echo "$(cat kubectl.sha256)  kubectl" | sha256sum --check
```

*Click to run on **kubernetes***

Expected output:

```
kubectl: OK                                                             shell
```

4. Install kubectl

```
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

*Click to run on **kubernetes***

5. Test kubectl

```
kubectl version --client
```

*Click to run on **kubernetes***

Expected output:

```
Client Version: v1.28.2                                                    shell
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
```

6. Test kubectl by retrieving the cluster state (we expect this to fail)

```
kubectl cluster-info
```

*Click to run on **kubernetes***

Expected output:

```
                                                                           shell
E1017 21:31:38.476094    1921 memcache.go:265] couldn't get current server API group
E1017 21:31:38.476522    1921 memcache.go:265] couldn't get current server API group
E1017 21:31:38.477836    1921 memcache.go:265] couldn't get current server API group
E1017 21:31:38.479106    1921 memcache.go:265] couldn't get current server API group
E1017 21:31:38.480359    1921 memcache.go:265] couldn't get current server API group

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
The connection to the server localhost:8080 was refused - did you specify the right h
```

Step 12: Kubernetes Installation: Create a Cluster with kubeadm

Create a cluster with kubeadm

1. Do a dryrun with kubeadm to create a cluster

```
sudo kubeadm init --cri-socket=unix:///run/containerd/containerd.sock --dry-run
```

*Click to run on **kubernetes***

2. If the dryrun is successful, continue with creating a cluster

```
sudo kubeadm init --cri-socket=unix:///run/containerd/containerd.sock
```

*Click to run on **kubernetes***

Expected output:

```
[init] Using Kubernetes version: v1.28.2                                   shell
[preflight] Running pre-flight checks
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your internet
[preflight] You can also perform this action in beforehand using 'kubeadm config imag
W1017 22:18:56.920986    4159 checks.go:835] detected that the sandbox image "registr
```

```
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names [ip-172-31-42-53 kubernetes ku
[certs] Generating "apiserver-kubelet-client" certificate and key
[certs] Generating "front-proxy-ca" certificate and key
[certs] Generating "front-proxy-client" certificate and key
[certs] Generating "etcd/ca" certificate and key
[certs] Generating "etcd/server" certificate and key
[certs] etcd/server serving cert is signed for DNS names [ip-172-31-42-53 localhost]
[certs] Generating "etcd/peer" certificate and key
[certs] etcd/peer serving cert is signed for DNS names [ip-172-31-42-53 localhost] ar
[certs] Generating "etcd/healthcheck-client" certificate and key
[certs] Generating "apiserver-etcd-client" certificate and key
[certs] Generating "sa" key and public key
[kubeconfig] Using kubeconfig folder "/etc/kubernetes"
[kubeconfig] Writing "admin.conf" kubeconfig file
[kubeconfig] Writing "kubelet.conf" kubeconfig file
[kubeconfig] Writing "controller-manager.conf" kubeconfig file
[kubeconfig] Writing "scheduler.conf" kubeconfig file
[etcd] Creating static Pod manifest for local etcd in "/etc/kubernetes/manifests"
[control-plane] Using manifest folder "/etc/kubernetes/manifests"
[control-plane] Creating static Pod manifest for "kube-apiserver"
[control-plane] Creating static Pod manifest for "kube-controller-manager"
[control-plane] Creating static Pod manifest for "kube-scheduler"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Starting the kubelet
[wait-control-plane] Waiting for the kubelet to boot up the control plane as static P
[apiclient] All control plane components are healthy after 9.002765 seconds
[upload-config] Storing the configuration used in ConfigMap "kubeadm-config" in the "
[kubelet] Creating a ConfigMap "kubelet-config" in namespace kube-system with the cor
[upload-certs] Skipping phase. Please see --upload-certs
[mark-control-plane] Marking the node ip-172-31-42-53 as control-plane by adding the
[mark-control-plane] Marking the node ip-172-31-42-53 as control-plane by adding the
[bootstrap-token] Using token: 0cy7s2.w9tjwwg2qdryn4aa
[bootstrap-token] Configuring bootstrap tokens, cluster-info ConfigMap, RBAC Roles
[bootstrap-token] Configured RBAC rules to allow Node Bootstrap tokens to get nodes
[bootstrap-token] Configured RBAC rules to allow Node Bootstrap tokens to post CSRs i
[bootstrap-token] Configured RBAC rules to allow the csrapprover controller automatic
[bootstrap-token] Configured RBAC rules to allow certificate rotation for all node cl
[bootstrap-token] Creating the "cluster-info" ConfigMap in the "kube-public" namespac
[kubelet-finalize] Updating "/etc/kubernetes/kubelet.conf" to point to a rotatable ku
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy
```

```
Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

Alternatively, if you are the root user, you can run:

  export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root

kubeadm join 172.31.42.53:6443 --token 0cy7s2.w9tjwwg2qdryn4aa \
    --discovery-token-ca-cert-hash sha256:b3b8cce7e14bab053afd42e4c5260370c90e86ccb60
```

2. Follow the instructions from the end of `kubeadm init`, this copies the kubeconfig file to where kubectl expectes the kubeconfig file to be. The kubeconfig file contains how you will authenticate to the kubernetes cluster.

```
mkdir -p $HOME/.kube
```

*Click to run on **kubernetes***

```
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
```

*Click to run on **kubernetes***

```
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

*Click to run on **kubernetes***

3. Test kubectl can retrieve the cluster info

```
kubectl cluster-info
```

*Click to run on **kubernetes***

Expected output:

```
Kubernetes control plane is running at https://172.31.42.53:6443                shell
CoreDNS is running at https://172.31.42.53:6443/api/v1/namespaces/kube-system/service

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

4. By default, control plane nodes are tainted Remove the taint on the node

```
kubectl taint nodes --all node-role.kubernetes.io/control-plane-
```

*Click to run on **kubernetes***

Expected output:

```
node/${vminfo:kubernetes:hostname} untainted
```

5. Check if the cluster is ready

```
kubectl get nodes
```

*Click to run on **kubernetes***

Expected output:

```
NAME                STATUS     ROLES           AGE   VERSION                    shell
${vminfo:kubernetes:hostname}   NotReady   control-plane   12m   v1.28.2
```

Step 13: Cilium

Install Cilium

1. Install the latest Cilium CLI

```
CILIUM_CLI_VERSION=$(curl -s https://raw.githubusercontent.com/cilium/cilium-
cli/main/stable.txt)
CLI_ARCH=amd64
if [ "$(uname -m)" = "aarch64" ]; then CLI_ARCH=arm64; fi
curl -L --fail --remote-name-all https://github.com/cilium/cilium-cli/releases
/download/${CILIUM_CLI_VERSION}/cilium-linux-${CLI_ARCH}.tar.gz{,.sha256sum}
sha256sum --check cilium-linux-${CLI_ARCH}.tar.gz.sha256sum
sudo tar xzvfC cilium-linux-${CLI_ARCH}.tar.gz /usr/local/bin
rm cilium-linux-${CLI_ARCH}.tar.gz{,.sha256sum}
```

*Click to run on **kubernetes***

Expected output

```
                                                                              shell
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 34.5M  100 34.5M    0     0  28.6M      0  0:00:01  0:00:01 --:--:-- 97.0M
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100    92  100    92    0     0    305      0 --:--:-- --:--:-- --:--:--   305
cilium-linux-amd64.tar.gz: OK
cilium
```

2. Install Cilium into the Kubernetes cluster pointed to by your current kubectl context:

```
cilium install --version 1.14.2
```

*Click to run on **kubernetes***

Expected output:

```
ℹ️  Using Cilium version 1.14.2                                    shell
🔮 Auto-detected cluster name: kubernetes
🔮 Auto-detected kube-proxy has been installed
```

3. Validate the Cilium install

```
cilium status --wait
```

*Click to run on **kubernetes***

Expected output:

```
    /¯¯\                                                           shell
 /¯¯\__/¯¯\    Cilium:             OK
 \__/¯¯\__/    Operator:           OK
 /¯¯\__/¯¯\    Envoy DaemonSet:    disabled (using embedded mode)
 \__/¯¯\__/    Hubble Relay:       disabled
    \__/       ClusterMesh:        disabled

Deployment             cilium-operator    Desired: 1, Ready: 1/1, Available: 1/1
DaemonSet              cilium             Desired: 1, Ready: 1/1, Available: 1/1
Containers:            cilium             Running: 1
                       cilium-operator    Running: 1
Cluster Pods:          2/2 managed by Cilium
Helm chart version:    1.14.2
Image versions         cilium             quay.io/cilium/cilium:v1.14.2@sha256:6263f3
                       cilium-operator    quay.io/cilium/operator-generic:v1.14.2@sha
```

4. Test the cluster's readiness

```
kubectl get nodes
```

*Click to run on **kubernetes***

```
NAME                STATUS   ROLES           AGE   VERSION                    shell
${vminfo:kubernetes:hostname}   Ready    control-plane   25m   v1.28.2
```

5. Check the Kubernetes components running as Pods

```
kubectl get pods -n kube-system
```

*Click to run on **kubernetes***

Expected output:

```
NAME                                READY   STATUS    RESTARTS   AGE     shell
cilium-operator-5d47789fcb-mft24    1/1     Running   0          31m
```

```
cilium-pngps                              1/1     Running   0          31m
coredns-5dd5756b68-qq55f                  1/1     Running   0          48m
coredns-5dd5756b68-qsj62                  1/1     Running   0          48m
etcd-ip-172-31-42-53                      1/1     Running   0          48m
kube-apiserver-ip-172-31-42-53            1/1     Running   0          48m
kube-controller-manager-ip-172-31-42-53   1/1     Running   0          48m
kube-proxy-lhz4p                          1/1     Running   0          48m
kube-scheduler-ip-172-31-42-53            1/1     Running   0          48m
```

Kubernetes uses etcd as its key-value store used for all cluster data. etcd is a CNCF graduated project. Kubernetes uses CoreDNS which is a CNCF graduated project. CoreDNS is a DNS server that chains plugins and serves as the cluster DNS.

Step 14: Kubernetes Basics

Kubernetes Basics

With our Kubernetes cluster ready, let's deploy basic Kubernetes resources

Namespace

1. A namespace isolates a group of resources within a Kubernetes cluster. Create a new namespace `my-namespace` .

```
kubectl create namespace my-namespace
```

*Click to run on **kubernetes***

Expected output:

```
namespace/my-namespace created
```

Pod

A Pod is the smallest deployable unit in Kubernetes. A Pod can be one or more containers that work together.

2. Use an imperative command to deploy an nginx webserver:

```
kubectl run nginx --image=nginx:1.25.3 --port=80 -n my-namespace
```

*Click to run on **kubernetes***

Expected output:

```
pod/nginx created                                                    shell
```

3. Find the Pod's virtual IP and test access to the nginx container.

```
kubectl get pods -o wide -n my-namespace
```

*Click to run on **kubernetes***

Expected output:

```
                                                                    shell
NAME    READY   STATUS    RESTARTS   AGE   IP          NODE              NOMINATED NO
nginx   1/1     Running   0          83s   <VIP>       <hostname>   <none>
```

4. Curl the nginx container on the Virtual IP, the command below is not click-to-run since you have to enter the Pod's VIP.

```
curl -sk http://<VIP> | head -n 4
```

Expected output:

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
```

5. Delete the Pod as we will deploy multiple replicas of the same Pod in the next step.

```
kubectl delete pod nginx -n my-namespace
```

*Click to run on **kubernetes***

Expected output:

```
pod "nginx" deleted                                                 shell
```

In the next several steps we'll look at how to create resources to scale an application and expose it outside the Kubernetes cluster.

Deployment

A Deployment is a Kubernetes resource that provides declartive updates at a controlled rate for Pods. This time you will use a Kubernetes manifest to deploy a Deployment.

6. Create a Kubernetes manifest for a Deployment in the `my-namespace` namespace for two replicas of Pod that runs the nginx webserver.

This time give the `app=nginx` labels to the Deployment and Pods of the Deployment so we can group and identify the Pods in a later step.

```
mkdir -p ~/manifests
cat <<EOF > ~/manifests/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
  name: nginx
  namespace: my-namespace
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.25.3
        ports:
        - containerPort: 80
EOF
```

*Click to run on **kubernetes***

7. Use `kubectl apply` to apply the configuration from the deployment.yaml manifest.

```
kubectl apply -f ~/manifests/deployment.yaml
```

*Click to run on **kubernetes***

Expected output:

```
deployment.apps/nginx created
```
shell

8. Check the status of the Deployment.

```
kubectl get deployment -n my-namespace
```

*Click to run on **kubernetes***

Expected output:

```
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
nginx   2/2     2            2           30s
```
shell

9. Check the Pods of the deployment.

```
kubectl get pods -n my-namespace -o wide
```

*Click to run on **kubernetes***

With 2 replicas of the nginx webserver running, let's expose the Pods of the Deployment to outside of the cluster

Services

A Service abstracts a group of Pods over a network using a label selector identify the Pods to expose.

There are multiple types of Kubernetes Services, we will use the NodePort service to expose the Service on each Node's IP of a cluster.

By default, a NodePort exposes the Service in the port range of 30000 to 32767.

Expose the Pods of the nginx Deployment to be reachable with your browser.

10. Create a manifest for a Service that uses the Pod's label in the `my-namespace` namespace.

```
cat <<EOF > ~/manifests/service.yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
  namespace: my-namespace
spec:
  type: NodePort
  selector:
    app: nginx
  ports:
    - port: 80
      nodePort: 30000
EOF
```

*Click to run on **kubernetes***

11. Apply the service.yaml manifest to create the Service.

```
kubectl apply -f ~/manifests/service.yaml
```

*Click to run on **kubernetes***

Expected output:

```
service/nginx-service created                                    shell
```

12. Check the Service was created.

```
kubectl get service -n my-namespace
```

*Click to run on **kubernetes***

13. Use your browser to test access to the nginx webserver http://kubernetes.${vminfo:kubernetes:public_ip}.sslip.io:30000.

You should see `Welcome to nginx~` .

At the end of this step, you've successfully deployed 2 replicas of an application container with a Pod and Deployment and exposed the application with a Service type NodePort.

Step 15: Helm

Helm

Helm is Kubernetes package manager and a CNCF graduated project. Helm helps you manage applications.
Helm charts are the packaging format. A chart are files in a particulary directory structure that describe related Kubernetes resources.
A chart can be used to deploy something simple like a single Pod with a single container or complex like an application composed of an HTTP server, database, caches, etc.
Helm charts are stored in Helm repositories.
You will use Helm to install several CNCF graduated projects later in this scenario.
1. Install Helm.

```
curl https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 | bash
```

*Click to run on **kubernetes***

Expected output:

```
                                                                 shell
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 11661  100 11661    0     0  87773      0 --:--:-- --:--:-- --:--:-- 88340
Downloading https://get.helm.sh/helm-v3.13.1-linux-amd64.tar.gz
Verifying checksum... Done.
Preparing to install helm into /usr/local/bin
helm installed into /usr/local/bin/helm
```

2. Verify Helm install.

```
helm version --client
```

*Click to run on **kubernetes***

Expected output:

```
                                                                 shell
version.BuildInfo{Version:"v3.13.1", GitCommit:"3547a4b5bf5edb5478ce352e18858d8a552a4
```

Step 16: Observability: Metrics

Observability: Metrics

There are 3 pillars of observability: metrics, traces, and logs. Some add a fourth pillar, events.
In this tutorial, you will install monitoring and logging solutions. Let's start with monitoring.

Prometheus stores and collects metrics as time series data. Prometheus is a monitoring and alerting toolkit and is a CNCF graduated project.

There are multiple ways to install Prometheus. For this tutorial, we will install Prometheus via Helm.

The kube-prometheus-stack is a community-maintained helm chart that is a "batteries-included" chart that includes the following:

- Prometheus Operator
- Prometheus with Prometheus rules
- Alertmanager
- Prometheus node-exporter
- Prometheus Adapter for Kubernetes Metrics APIs
- kube-state-metrics
- Grafana with dashboards

1. Add the Prometheus community helm repository.

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
```

*Click to run on **kubernetes***

Expected output:

```
"prometheus-community" has been added to your repositories
```
shell

2. Update information of available charts locally from chart repositories.

```
helm repo update
```

*Click to run on **kubernetes***

Expected output:

```
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "prometheus-community" chart repository
Update Complete. ⎈Happy Helming!⎈
```
shell

3. Use helm to install the kube-prometheus-stack

```
helm install kube-prometheus-stack prometheus-community/kube-prometheus-stack \
--set alertmanager.service.type=NodePort \
--set prometheusOperator.service.type=NodePort \
--set prometheus.service.type=NodePort \
--set grafana.service.type=NodePort \
--set grafana.service.nodePort=30140 \
--namespace kube-prometheus-stack \
--create-namespace \
--set namespaceOverride=kube-prometheus-stack \
--set grafana.namespaceOverride=kube-prometheus-stack \
--set kube-state-metrics.namespaceOverride=kube-prometheus-stack \
```

```
--set prometheus-node-exporter.namespaceOverride=kube-prometheus-stack
```

*Click to run on **kubernetes***

Expected output:

```
NAME: kube-prometheus-stack                                                    shell
LAST DEPLOYED: Wed Oct 18 21:31:24 2023
NAMESPACE: kube-prometheus-stack
STATUS: deployed
REVISION: 1
NOTES:
kube-prometheus-stack has been installed. Check its status by running:
  kubectl --namespace kube-prometheus-stack get pods -l "release=kube-prometheus-stac

Visit https://github.com/prometheus-operator/kube-prometheus for instructions on how
```

4. Check the status of the installation.

```
kubectl --namespace kube-prometheus-stack get pods -l "release=kube-prometheus-
stack"
```

*Click to run on **kubernetes***

Expected output:

```
                                                                               shell
NAME                                                   READY   STATUS    RESTART
kube-prometheus-stack-kube-state-metrics-6bdd78dc74-4frg8   1/1     Running   0
kube-prometheus-stack-operator-67bc68d75-h8z5d              1/1     Running   0
kube-prometheus-stack-prometheus-node-exporter-nhpzn        1/1     Running   0
```

5. Kubernetes Services expose applications. A Service type of NodePort exposes Services at each Kubernetes nodes' IP at a static port from 30000 to 32767. Check the services

```
kubectl get services -n kube-prometheus-stack
```

*Click to run on **kubernetes***

Expected output:

```
                                                                               shell
NAME                                          TYPE        CLUSTER-IP      EXTERNAL
alertmanager-operated                         ClusterIP   None            <none>
kube-prometheus-stack-alertmanager            NodePort    10.102.162.47   <none>
kube-prometheus-stack-grafana                 NodePort    10.110.209.69   <none>
kube-prometheus-stack-kube-state-metrics      ClusterIP   10.107.28.248   <none>
kube-prometheus-stack-operator                NodePort    10.111.61.81    <none>
kube-prometheus-stack-prometheus              NodePort    10.96.184.91    <none>
kube-prometheus-stack-prometheus-node-exporter   ClusterIP   10.97.91.228    <none>
prometheus-operated                           ClusterIP   None            <none>
```

6.

To access Prometheus go to your node's IP on port 30090,
http://kubernetes.${vminfo:kubernetes:public_ip}.sslip.io:30090
7.
Prometheus uses Prometheus Querying Language (PromQL) to query metrics. Run a few PromQL queries in the browers: `node_cpu_seconds_total{cpu="0"}[5m]` shows the total amount of CPU time spent over the last 5 minutes
8.
To access Grafana go to your node's IP on port 30140,
http://kubernetes.${vminfo:kubernetes:public_ip}.sslip.io:30140 and use the default admin credentials:
username = `admin` , password = `prom-operator`
9.
From the hamburger menu in the top left, go to Dashboards and checkout a dashboard like the `Kubernetes/Kubelet` dashboard

Step 17: Observability: Logging

Logging

You can gather logs from containers with the `kubectl -n <namespace> logs <pod name> -c <container name if multiple containers in pod>` command.
An application may use multiple containers and it'll be difficult to obtain logs through single containers. A tool to collect logs of an application will simplify log collection.
Fluentd is an open source log aggregator, collector, processor written in Ruby and is a CNCF graduated project.
Fluent Bit is an open source log aggregator, collector, processor, forwarder written in C. Fluent Bit is a CNCF graduated sub-project under the umbrella of Fluentd.
Both Fluentd and Fluent Bit use plugins to integrate with data sources and data outputs.
Fluent Bit excels in highly distributed environment and has a smaller footprint than FluentD ~450kb vs ~40MB of memory. Both were developed by Treasure Data.
Fluent Bit is not as flexible as Fluentd. Fluent Bit has about 45 plugins versus Fluentd has about 700 plugins.
Fluent Bit has native support for environmental metric collection.
The use of both Fluent Bit and Fluentd optimizes resource consumption on the nodes with Fluent Bit's smaller footprint and optimizes the flexibilty of Fluentd with more than 700 plugins available.
For this tutorial, we'll use Fluentd to collect logs

Install Elasticsearch, Logstash, and Kibana

We will first install Elasticsearch, Logstash, and Kibana aka the ELK stack as the logging backend for fluentd to send logs to.

We will install the following:

- Elasticsearch which is a search/analytics engine and a log analytics tool
- Logstash to collect logs and send to Elasticsearch for storage and analysis
- Filebeat to
- Kibana to visualize the data stored in Elasticsearch

1. Add the elastic helm repo.

```
helm repo add elastic https://helm.elastic.co
helm repo update
```

*Click to run on **kubernetes***

Expected output:

```
"elastic" has been added to your repositories                                    shell
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "fluent" chart repository
...Successfully got an update from the "elastic" chart repository
...Successfully got an update from the "prometheus-community" chart repository
Update Complete. ⎈Happy Helming!⎈
```

2. Install Elasticsearch.

```
helm install elasticsearch elastic/elasticsearch --set replicas=1 --set
minimumMasterNodes=1 --set secret.password=changeme --set
resources.requests.cpu=100m --set resources.requests.memory=100Mi --set
persistence.enabled=false -n elk --create-namespace
```

*Click to run on **kubernetes***

Expected output:

```
NAME: elasticsearch                                                              shell
LAST DEPLOYED: Thu Nov  2 04:28:58 2023
NAMESPACE: elk
STATUS: deployed
REVISION: 1
NOTES:
1. Watch all cluster members come up.
  $ kubectl get pods --namespace=elk -l app=elasticsearch-master -w
2. Retrieve elastic user's password.
  $ kubectl get secrets --namespace=elk elasticsearch-master-credentials -ojsonpath=
3. Test cluster health using Helm test.
  $ helm --namespace=elk test elasticsearch
```

Elasticsearch will take a few minutes to be at a `ready` state/ You can check the status of the Elasticsearch pod with `kubectl get pods`

```
kubectl get pods -n elk
```

*Click to run on **kubernetes***

Expected output:

```
NAME                     READY   STATUS    RESTARTS   AGE                    shell
elasticsearch-master-0   1/1     Running   0          2m19s
```

Wait until the `elasticsearch-master-0` is ready before proceeding.

3. Install Logstash

```
helm install logstash elastic/logstash -n elk
```

*Click to run on **kubernetes***

Expected output:

```
NAME: logstash                                                              shell
LAST DEPLOYED: Thu Nov  2 05:42:26 2023
NAMESPACE: elk
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
1. Watch all cluster members come up.
  $ kubectl get pods --namespace=elk -l app=logstash-logstash -w
```

Wait until the logstash Pod is ready before proceeding.

```
kubectl get pods --namespace=elk -l app=logstash-logstash
```

*Click to run on **kubernetes***

Expected output:

```
NAME                  READY   STATUS    RESTARTS   AGE                     shell
logstash-logstash-0   1/1     Running   0          94s
```

4. Install Kibana.

```
helm install kibana elastic/kibana --set resources.requests.cpu=100m --set
resources.requests.memory=100Mi --set service.type=NodePort --set
service.nodePort=30001 -n elk
```

*Click to run on **kubernetes***

Expected output:

```
NAME: kibana                                                                shell
LAST DEPLOYED: Thu Nov  2 05:48:14 2023
NAMESPACE: elk
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
1. Watch all containers come up.
```

```
  $ kubectl get pods --namespace=elk -l release=kibana -w
2. Retrieve the elastic user's password.
  $ kubectl get secrets --namespace=elk elasticsearch-master-credentials -ojsonpath=
3. Retrieve the kibana service account token.
  $ kubectl get secrets --namespace=elk kibana-kibana-es-token -ojsonpath='{.data.tol
```

Wait until the kibana Pod is ready before proceeding

```
kubectl get pods -n elk -l app=kibana
```

*Click to run on **kubernetes***

Expected output:

```
NAME                              READY   STATUS    RESTARTS   AGE             shell
kibana-kibana-6888db469c-djkv7    1/1     Running   0          111s
```

Use your browser to access Kibana https://kubernetes.${vminfo:kubernetes:public_ip}.sslip.io:30001

Login in with the username = `elastic` , password = `changeme`

fluentd

5. Create the required ServiceAccount, ClusterRole, ClusterRoleBinding for fluentd.

```
cat <<EOF > ~/manifests/fluentd-rbac.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: fluentd
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: fluentd
  namespace: kube-system
rules:
- apiGroups: [""]
  resources:
  - pods
  - namespaces
  verbs:
  - get
  - list
  - watch
---
```

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: fluentd
roleRef:
  kind: ClusterRole
  name: fluentd
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: ServiceAccount
  name: fluentd
  namespace: kube-system
EOF
```

*Click to run on **kubernetes***

6. Apply the manifest.

```
kubectl apply -f ~/manifests/fluentd-rbac.yaml
```

*Click to run on **kubernetes***

Expected output:

```
serviceaccount/fluentd created                                    shell
clusterrole.rbac.authorization.k8s.io/fluentd created
clusterrolebinding.rbac.authorization.k8s.io/fluentd created
```

7. Create the manifest for the fluentd DaemonSet.

```
cat <<EOF > ~/manifests/fluentd-ds.yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd
  namespace: kube-system
  labels:
    k8s-app: fluentd-logging
    version: v1
spec:
  selector:
    matchLabels:
      k8s-app: fluentd-logging
      version: v1
  template:
    metadata:
```

```
    labels:
      k8s-app: fluentd-logging
      version: v1
  spec:
    serviceAccount: fluentd
    serviceAccountName: fluentd
    tolerations:
    - key: node-role.kubernetes.io/master
      effect: NoSchedule
    containers:
    - name: fluentd
      image: fluent/fluentd-kubernetes-daemonset:elasticsearch
      env:
      - name:  FLUENT_ELASTICSEARCH_HOST
        value: "elasticsearch-master.elk.svc.cluster.local"
      - name:  FLUENT_ELASTICSEARCH_PORT
        value: "9200"
      - name: FLUENT_ELASTICSEARCH_SCHEME
        value: "http"
      - name: FLUENT_UID
        value: "0"
      # X-Pack Authentication
      # =====================
      - name: FLUENT_ELASTICSEARCH_USER
        value: "elastic"
      - name: FLUENT_ELASTICSEARCH_PASSWORD
        value: "changeme"
      resources:
        limits:
          memory: 200Mi
        requests:
          cpu: 100m
          memory: 200Mi
      volumeMounts:
      - name: varlog
        mountPath: /var/log
      - name: varlibdockercontainers
        mountPath: /var/lib/docker/containers
        readOnly: true
    terminationGracePeriodSeconds: 30
```

```
      volumes:
      - name: varlog
        hostPath:
          path: /var/log
      - name: varlibdockercontainers
        hostPath:
          path: /var/lib/docker/containers
EOF
```

*Click to run on **kubernetes***

8. Apply the manifest.

```
kubectl apply -f ~/manifests/fluentd-ds.yaml
```

*Click to run on **kubernetes***

Expected output:

```
daemonset.apps/fluentd created                                    shell
```

9. Navigate to the kibana UI > Management > Index Management to see a new Logstash index generated by the fluentd DaemonSet.

Step 18: Policy: OPA and Gatekeeper

Open Policy Agent and Gatekeeper

Open Policy Agent (OPA) is a policy engine with policy-as-code and a CNCF graduated project. OPA is used to enforce policies in cloud native environments.

OPA Gatekeeper is a customizable admission webhook that enforces policies executed by OPA. With Gatekeeper, users can customize admission control configuration easier.

With OPA Gatekeeper, you can enforce policies like container images must be pulled from a specific container registry or not allow privilege escalation in containers.

1. Add the Gatekeeper helm repo.

```
helm repo add gatekeeper https://open-policy-agent.github.io/gatekeeper/charts
helm repo update
```

*Click to run on **kubernetes***

Expected output:

```
"gatekeeper" has been added to your repositories                  shell
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "gatekeeper" chart repository
...Successfully got an update from the "fluent" chart repository
```

```
...Successfully got an update from the "elastic" chart repository
...Successfully got an update from the "prometheus-community" chart repository
Update Complete. ⎈Happy Helming!⎈
```

2. Install Gatekeeper.

```
helm install gatekeeper/gatekeeper --name-template=gatekeeper --namespace
gatekeeper-system --create-namespace
```

*Click to run on **kubernetes***

Expected output:

```
NAME: gatekeeper                                                    shell
LAST DEPLOYED: Mon Nov  6 03:51:16 2023
NAMESPACE: gatekeeper-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

Constraint Framework

A Constraint is the declaration of a policy. For example, if you require each resource in a specific namespace requires a **key** label.

Before you can use Constraints, you must have a Constraint Template. The Constraint Template defines the input parameters and the rego that defines and enforces a policy.

3. Create a Constraint Template that requires an **app** label.

```
cat <<EOF | kubectl apply -f -
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8srequiredlabels
spec:
  crd:
    spec:
      names:
        kind: K8sRequiredLabels
      validation:
        openAPIV3Schema:
          type: object
          properties:
            labels:
              type: array
              items:
```

```
              type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8srequiredlabels

        violation[{"msg": msg, "details": {"missing_labels": missing}}] {
          provided := {label | input.review.object.metadata.labels[label]}
          required := {label | label := input.parameters.labels[_]}
          missing := required - provided
          count(missing) > 0
          msg := sprintf("you must provide labels: %v", [missing])
        }
EOF
```

*Click to run on **kubernetes***

Expected output:

```
constrainttemplate.templates.gatekeeper.sh/k8srequiredlabels created
```
shell

4. Create a Constraint that enforces the ConstraintTemplate to require a label. Specify this policy for Namespaces with the label `owner`. Note this is a dry-run.

```
cat <<EOF | kubectl apply -f -
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sRequiredLabels
metadata:
  name: ns-must-have-owner
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Namespace"]
  parameters:
    labels: ["owner"]
EOF
```

*Click to run on **kubernetes***

```
k8srequiredlabels.constraints.gatekeeper.sh/ns-must-have-owner created
```
shell

5. Test the Constraint.

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
```

```
kind: Namespace
metadata:
  name: new-namespace
  labels:
    owner: you
EOF
```

*Click to run on **kubernetes***

Expected output:

```
namespace/new-namespace created                                    shell
```

6. Test the Constraint with a Namespace without an `owner` label.

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Namespace
metadata:
  name: rejected-namespace
EOF
```

*Click to run on **kubernetes***

Expected output:

```
                                                                   shell
Error from server (Forbidden): error when creating "STDIN": admission webhook "valido
```